

IEEE
Std 610.12-1990
(Revision and redesignation of
IEEE Std 792-1983)

IEEE Standard Glossary of Software Engineering Terminology

Sponsor
**Standards Coordinating Committee
of the
Computer Society of the IEEE**

Approved September 28, 1990
IEEE Standards Board

Abstract: IEEE Std 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, identifies terms currently in use in the field of Software Engineering. Standard definitions for those terms are established.
Keywords: Software engineering; glossary; terminology; definitions; dictionary

ISBN 1-55937-067-X

Copyright © 1990 by

**The Institute of Electrical and Electronics Engineers
345 East 47th Street, New York, NY 10017, USA**

*No part of this document may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.*

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least once every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Foreword

(This Foreword is not a part of IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.)

The computer field is continuing to expand. New terms are being generated and new meanings are being adopted for existing terms. The IEEE Computer Dictionary project was undertaken to document this vocabulary. Its purpose is to identify terms currently in use in the computer field and to establish standard definitions for these terms. The dictionary is intended to serve as a useful reference for those in the computer field and for those who come into contact with computers either through their work or in their everyday lives.

The dictionary is being developed as a set of subject-area glossaries covering Computer Architecture, Computer Processors, Computer Storage, Software Engineering, Mathematics of Computing, Theory of Computation, Computer Applications, Artificial Intelligence, Data Management, Image Processing and Pattern Recognition, Modeling and Simulation, Computer Graphics, Computer Networking, Computer Languages, and Computer Security and Privacy. This glossary contains the terms related to Software Engineering. It updates IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology (ANSI).

Every effort has been made to use definitions from established standards in this dictionary. When existing standards were found to be incomplete, unclear, or inconsistent with other entries in the dictionary, however, new, revised, or composite definitions have been developed.

At the time this glossary was approved, the following people formed the steering committee of the Computer Dictionary working group:

Jane Radatz, Chairperson, *Software Engineering Glossary*

Other subgroup leaders:

Anne Geraci
Freny Katki
Dr. John Lane

Louise McMonegal
Bennett Meyer
Dr. Hugh Porteous
Dr. Fredrick Springsteel

Paul Wilson
Mary Yee
John Young

Other working group members who contributed to this glossary were as follows:

Russell J. Abbott
A. Frank Ackerman
Roger R. Baldwin
H. Ronald Berlack
J. David Bezek
James H. Bradley
Kathleen L. Briggs
Homer C. Carney
Susann Chonoles
Taz Daughtrey
Frank J. Douglas
William P. Dupras

John D. Earls
Mary Forcht-Tucker
David Gelperin
Al Gillen
Shirley A. Gloss-Soler
John A. Goetz
David A. Gustafson
Virl Haas
James Ingram
Gary S. Lindsay
Robert McBeth
Alicia McCurdy

Dr. José Muñoz
Geraldine Neidhart
Mary Rasmussen
Max Schindler
Paul Schmid
Leonard W. Seagren
Sonja Peterson Shields
Kevin Smith
Wayne Smith
Paul U. Thompson
Andrew H. Weigel
W. Martin Wong

Special representatives to the Computer Dictionary working group were as follows:

Frank Jay, Advisor, *IEEE Standards Office*

Rollin Mayer, Liaison, *Accredited Standards Committee X3K5*

The following organizations supported employee participation in the development of this standard:

Applied Information Development
Atlantic Consultants
Babcock and Wilcox
Burroughs Wellcome
Carney Associates
Computer Sciences Corporation
Datapoint Corporation
Dutchess Engineering Company
Edinboro University of Pennsylvania
Electronics Design Magazine
Eyring Research Institute
General Electric Company
Harris Corporation
Information Spectrum, Inc.
Institute for Zero Defect Software
International Bureau of Software Test
Kansas State University

Lockheed
Logicon
Marine Midland Bank
The MITRE Corporation
Perkin-Elmer Corporation
Quality Assurance Institute
Rabbitt Software Corporation
RCA
Sanders Associates
SILOGIC
Softtran
Teledyne Brown Engineering
University of Wisconsin, Madison
U.S. Naval Facilities
U.S. Dept. of HUD
Wyse Technology

The IEEE 610 working group wishes to acknowledge the contribution of those who developed IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology (ANSI), which formed the basis for this glossary. The steering committee of this group had the following members:

Shirley A. Gloss-Soler, *Chair*

Russell J. Abbott
Joan P. Bateman
Stephen R. Beason
Milton E. Boyd, Jr.
Kurt F. Fischer

John M. Ives
John J. McKissick, Jr.
Albrecht J. Neumann
John N. Postak

Jane W. Radatz
Marilyn J. Stewart
Alan N. Sukert
Donald A. Woodmancy
David Yablon



The sponsor for the Computer Dictionary project is the IEEE Computer Society Standards Coordinating Committee, which balloted this document for submission to the IEEE Standards Board. At the time this standard was approved, the committee had the following membership:

Harrison Beasley	Chris Haynes	Louis Miller
H. Ronald Berlack	Rick Henderson	James Mollenauer
William Billowitch	Ken Hobday	Jim Mooney
Richard Boberg	Scott Hopkinson	Gary A. Nelson
John Boebinger	John Horch	Tom Pittman
Paul L. Borrill	Russell Housley	Robert M. Poston
Terry Bowen	Charles Hudson	Shlomo Pri-Tal
Elliott Brebner	Marlyn Huckleby	Jane Radatz
J. Reese Brown, Jr.	Mike Humphrey	Michael Raynham
Lin Brown	John Hyde	Gordon Robinson
Fletcher Buckley	James Isaak	Larry Saunders
Randy Bush	David James	Richard Schmidt
Clyde Camp	Hal Jespersen	Norman Schneidewind
Steve Carter	Richard Kalish	Rudolph Schubert
Alan Cobb	Matt Kaltenbach	David Schultz
Paul Cook	Hans Karlsson	Karen Sheaffer
Bill Corwin	Freni Katki	Basil Sherlund
Alan Davis	Guy Kelley	Sava Sherr
Steven Deller	Kim Kirkpatrick	Fred Springsteel
Bulent Dervisoglu	Bob Knight	John Starkweather
Bob Donnan	Stanley Krolikoski	Dennis Steinauer
Paul Eastman	John B. Lane	Robert Sulgrove
D. Vera Edelstein	Ron Leckie	Oryal Tanir
Tim Elsmore	Kevin Lewis	Michael D. Teener
Dick Evans	William Lidinsky	Donn Terry
Richard Fairley	Donald C. Loughry	Pat Thaler
Wayne Fischer	Al Lowenstein	Joseph Toy
Kester Fong	Bill Maciejewski	Leonard Tripp
David Gelperin	Roger Martin	Margaret Updike
Anne Geraci	Philip Marriott	Eike Waltz
Al Gilman	Colin Maunder	John W. Walz
John Graham	John McGrory	Camille White-Partain
Steve Grout	Louise McMonegal	Les Wibberley
Dave Gustavson	Sunil Mehta	Cynthia Wright
Al Hankinson	Paul Menchini	John Young
Fred Harrison	Jerry Mersky	Jason Zions
	Bennett Meyer	

When the IEEE Standards Board approved this standard on September 28, 1990, it had the following membership:

Marco W. Migliaro, *Chairman*

James M. Daly, *Vice Chairman*

Andrew G. Salem, *Secretary*

Dennis Bodson
Paul L. Borrill
Fletcher J. Buckley
Allen L. Clapp
Stephen R. Dillon
Donald C. Fleckenstein
Jay Forster*
Thomas L. Hannan

Kenneth D. Hendrix
John W. Horch
Joseph L. Koepfinger*
Irving Kolodny
Michael A. Lawler
Donald J. Loughry
John E. May, Jr.

Lawrence V. McCall
L. Bruce McClung
Donald T. Michael*
Stig Nilsson
Roy T. Oishi
Gary S. Robinson
Terrance R. Whittemore
Donald W. Zipse

*Member Emeritus

Contents

SECTION	PAGE
1. Scope.....	7
2. Glossary Structure.....	7
3. Definitions for Software Engineering Terms.....	7
4. Bibliography	82

FIGURES

Fig 1	Block Diagram	13
Fig 2	Box Diagram.....	13
Fig 3	Bubble Chart.....	14
Fig 4	Call Graph.....	15
Fig 5	Case Construct	15
Fig 6	Data Flow Diagram	24
Fig 7	Data Structure Diagram	24
Fig 8	Directed Graph.....	27
Fig 9	Documentation Tree	28
Fig 10	Flowchart.....	33
Fig 11	Graph (1).....	36
Fig 12	Graph (2).....	36
Fig 13	If-Then-Else Construct	38
Fig 14	Input-Process-Output Chart.....	40
Fig 15	Sample Software Life Cycle	68
Fig 16	Structure Chart.....	71
Fig 17	UNTIL Construct.....	79
Fig 18	WHILE Construct	82

IEEE Standard Glossary of Software Engineering Terminology

1. Scope

This glossary defines terms in the field of Software Engineering. Topics covered include addressing; assembling, compiling, linking, loading; computer performance evaluation; configuration management; data types; errors, faults, and failures; evaluation techniques; instruction types; language types; libraries; microprogramming; operating systems; quality attributes; software documentation; software and system testing; software architecture; software development process; software development techniques; and software tools.

Every effort has been made to include all terms that meet these criteria. Terms were excluded if they were considered to be parochial to one group or organization; company proprietary or trademarked; multi-word terms whose meaning could be inferred from the definitions of the component words; or terms whose meaning in the computer field could be directly inferred from their standard English meaning.

This glossary is an update and expansion of IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology (ANSI) [3].¹ It increases the number of terms from approximately 500 to 1300, and updates or refines the definitions of many terms included in the initial glossary. A few terms that were included in the initial glossary have been moved to other glossaries in the 610 series. Some definitions have been recast in a system, rather than software, context. Every effort has been made to preserve the fine work that went into the initial glossary.

2. Glossary Structure

Entries in the glossary are arranged alphabetically. An entry may consist of a single word, such as “software,” a phrase, such as

¹Numbers in brackets correspond to those in the Bibliography in Section 4.

“test case,” or an acronym, such as “CM.” Phrases are given in their natural order (test plan) rather than in reversed order (plan, test).

Blanks precede all other characters in alphabetizing. Hyphens and slashes are treated as blanks. Alternative spellings are shown in parentheses.

If a term has more than one definition, the definitions are numbered. In most cases, noun definitions are given first, followed by verb and adjective definitions as applicable. Examples, notes, and illustrations have been added to clarify selected definitions.

The following cross-references are used to show a term's relationship to other terms in the dictionary:

- *Contrast with* refers to a term with an opposite or substantially different meaning.
- *Syn* refers to a synonymous term.
- *See also* refers to a related term.
- *See* refers to a preferred term or to a term where the desired definition can be found.

The word “deprecated” indicates a term or definition whose use is discouraged because such use is obsolete, misleading, or ambiguous. “DoD” refers to usage by the U.S. Department of Defense.

3. Definitions for Software Engineering Terms

1GL. Acronym for **first generation language**.
See: machine language.

2GL. Acronym for **second generation language**. *See: assembly language.*

3GL. Acronym for **third generation language**.
See: high order language.

4GL. Acronym for **fourth generation language**.

5GL. Acronym for **fifth generation language**.

abend. Abbreviation for **abnormal end**.

abnormal end (abend). Termination of a process prior to completion. *See also:* **abort**; **exception**.

abort. To terminate a process prior to completion. *See also:* **abend**; **exception**.

absolute address. An address that is permanently assigned to a device or storage location and that identifies the device or location without the need for translation or calculation. *Syn:* **explicit address**; **specific address**. *Contrast with:* **relative address**; **relocatable address**; **symbolic address**. *See also:* **absolute assembler**; **absolute code**; **absolute instruction**; **absolute loader**.

absolute assembler. An assembler that produces absolute code. *Contrast with:* **relocating assembler**.

absolute code. Code in which all addresses are absolute addresses. *Contrast with:* **relocatable code**. *Syn:* **specific code**.

absolute instruction. A computer instruction in which all addresses are absolute addresses. *See also:* **direct instruction**; **effective instruction**; **immediate instruction**; **indirect instruction**.

absolute loader. A loader that reads absolute machine code into main memory, beginning at the initial address assigned to the code by the assembler or compiler, and performs no address adjustments on the code. *Contrast with:* **relocating loader**.

abstract data type. A data type for which only the properties of the data and the operations to be performed on the data are specified, without concern for how the data will be represented or how the operations will be implemented.

abstraction. (1) A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information. *See also:* **data abstraction**.
(2) The process of formulating a view as in (1).

acceptance criteria. The criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity. *See also:* **requirement**; **test criteria**.

acceptance testing. (1) (IEEE Std 1012-1986 [12]) Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.
(2) Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component.
Contrast with: **development testing**. *See also:* **operational testing**; **qualification testing**.

accuracy. (1) A qualitative assessment of correctness, or freedom from error.
(2) A quantitative measure of the magnitude of error.
Contrast with: **precision**.

active redundancy. In fault tolerance, the use of redundant elements operating simultaneously to prevent, or permit recovery from, failures. *Contrast with:* **standby redundancy**.

actual instruction.* *See:* **effective instruction**.
* Deprecated.

actual parameter. *See:* **argument** (3).

Ada. *Note:* Ada and other specific computer languages are defined in P610.13 [17].

adaptability. *See:* **flexibility**.

adaptation data. Data used to adapt a program to a given installation site or to given conditions in its operational environment.

adaptation parameter. A variable that is given a specific value to adapt a program to a given installation site or to given conditions in its operational environment; for example, the variable `Installation_Site_Latitude`.

adaptive maintenance. Software maintenance performed to make a computer program usable in a changed environment.

Contrast with: **corrective maintenance**; **perfective maintenance**.

address. (1) A number, character, or group of characters that identifies a given device or storage location.

(2) To refer to a device or storage location by an identifying number, character, or group of characters.

See also: **absolute address**; **effective address**; **implied addressing**; **indirect address**; **relative address**; **relocatable address**; **symbolic address**; **virtual address**.

address field. Any of the fields of a computer instruction that contain addresses, information necessary to derive addresses, or values of operands. *Syn:* **address part**. *Contrast with:* **operation field**.

address format. (1) The number and arrangement of address fields in a computer instruction. *See also:* ***n*-address instruction**; ***n*-plus-one address instruction**.

(2) The number and arrangement of elements within an address, such as the elements needed to identify a particular channel, device, disk sector, and record in magnetic disk storage.

address modification. Any arithmetic, logical, or syntactic operation performed on an address. *See also:* **effective address**; **indexed address**; **relative address**; **relocatable address**.

address part. *See:* **address field**.

address space. (1) The addresses that a computer program can access. *Note:* In some systems, this may be the set of physical storage locations that a program can access, disjoint from other programs, together with the set of virtual addresses referring to those storage locations, which may be accessible by other programs.

(2) The number of memory locations that a central processing unit can address.

addressing exception. An exception that occurs when a program calculates an address outside the bounds of the storage available to it. *See also:* **data exception**; **operation exception**; **overflow exception**;

protection exception; **underflow exception**.

afferent. Pertaining to a flow of data or control from a subordinate module to a superordinate module in a software system. *Contrast with:* **efferent**.

algebraic language. A programming language that permits the construction of statements resembling algebraic expressions, such as $Y = X + 5$. For example, FORTRAN. *See also:* **algorithmic language**; **list processing language**; **logic programming language**.

algorithm. (1) A finite set of well-defined rules for the solution of a problem in a finite number of steps; for example, a complete specification of a sequence of arithmetic operations for evaluating $\sin x$ to a given precision.

(2) Any sequence of operations for performing a specific task.

algorithmic language. A programming language designed for expressing algorithms; for example, ALGOL. *See also:* **algebraic language**; **list processing language**; **logic programming language**.

allocated baseline. In configuration management, the initial approved specifications governing the development of configuration items that are part of a higher level configuration item. *Contrast with:* **developmental configuration**; **functional baseline**; **product baseline**. *See also:* **allocated configuration identification**.

allocated configuration identification. In configuration management, the current approved specifications governing the development of configuration items that are part of a higher level configuration item. Each specification defines the functional characteristics that are allocated from those of the higher level configuration item, establishes the tests required to demonstrate achievement of its allocated functional characteristics, delineates necessary interface requirements with other associated configuration items, and establishes design constraints, if any. *Contrast with:* **func-**

tional configuration identification; product configuration identification. *See also:* **allocated baseline.**

allocation. (1) The process of distributing requirements, resources, or other entities among the components of a system or program.

(2) The result of the distribution in (1).

anomaly. (IEEE Std 1012-1986 [12]) Anything observed in the documentation or operation of software that deviates from expectations based on previously verified software products or reference documents.

anticipatory buffering. A buffering technique in which data are stored in a buffer in anticipation of a need for the data. *See also:* **dynamic buffering; simple buffering.**

anticipatory paging. A storage allocation technique in which pages are transferred from auxiliary storage to main storage in anticipation of a need for those pages. *Contrast with:* **demand paging.**

application generator. A code generator that produces programs to solve one or more problems in a particular application area; for example, a payroll generator.

application-oriented language. A computer language with facilities or notations applicable primarily to a single application area; for example, a language for computer-assisted instruction or hardware design. *See also:* **authoring language; specification language; query language; simulation language.**

application software. Software designed to fulfill specific needs of a user; for example, software for navigation, payroll, or process control. *Contrast with:* **support software; system software.**

architectural design. (1) The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system. *See also:* **functional design.**

(2) The result of the process in (1).

architecture. The organizational structure of a system or component. *See also:* **component; module; subprogram; routine.**

argument. (1) An independent variable; for example, the variable m in the equation $E = mc^2$.

(2) A specific value of an independent variable; for example, the value $m = 24$ kg.

(3) A constant, variable, or expression used in a call to a software module to specify data or program elements to be passed to that module. *Syn:* **actual parameter.** *Contrast with:* **formal parameter.**

array. An n -dimensional ordered set of data items identified by a single name and one or more indices, so that each element of the set is individually addressable. For example, a matrix, table, or vector.

artificial intelligence. *Note:* P610.8 [15] defines terminology pertaining to artificial intelligence.

artificial language. *See:* **formal language.**

assemble. To translate a computer program expressed in an assembly language into its machine language equivalent. *Contrast with:* **compile; disassemble; interpret.**

assemble-and-go. An operating technique in which there are no stops between the assembling, linking, loading, and execution of a computer program.

assembled origin. The address of the initial storage location assigned to a computer program by an assembler, a compiler, or a linkage editor. *Contrast with:* **loaded origin.** *See also:* **offset (1); starting address.**

assembler. A computer program that translates programs expressed in assembly language into their machine language equivalents. *See also:* **absolute assembler; cross-assembler; relocating assembler.** *Contrast with:* **compiler; interpreter.**

assembler code. *See:* **assembly code.**

assembler language. *See:* **assembly language.**

assembly code. Computer instructions and data definitions expressed in a form that can be recognized and processed by an assembler. *Syn:* **assembler code**. *Contrast with:* **compiler code; interpretive code; machine code**.

assembly language. A programming language that corresponds closely to the instruction set of a given computer, allows symbolic naming of operations and addresses, and usually results in a one-to-one translation of program instructions into machine instructions. *Syn:* **assembler language; low level language; second generation language**. *Contrast with:* **fifth generation language; fourth generation language; high order language; machine language**. *Note:* Specific languages are defined in P610.13 [17].

assertion. A logical expression specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during program execution. Types include input assertion, loop assertion, output assertion. *See also:* **invariant; proof of correctness**.

assignment statement. A computer program statement that assigns a value to a variable; for example, $Y := X - 5$. *Contrast with:* **control statement; declaration**. *See also:* **clear; initialize; reset**.

atomic type. A data type, each of whose members consists of a single, nondecomposable data item. *Syn:* **primitive type**. *Contrast with:* **composite type**.

attribute. A characteristic of an item; for example, the item's color, size, or type. *See also:* **quality attribute**.

audit. An independent examination of a work product or set of work products to assess compliance with specifications, standards, contractual agreements, or other criteria. *See also:* **functional configuration audit; physical configuration audit**.

authoring language. A high level programming language used to develop courseware

for computer-assisted instruction. *See also:* **authoring system**.

authoring system. A programming system that incorporates an authoring language.

automated verification system. (1) A software tool that accepts as input a computer program and a representation of its specification and produces, possibly with human help, a proof or disproof of the correctness of the program. (2) Any software tool that automates part or all of the verification process.

availability. The degree to which a system or component is operational and accessible when required for use. Often expressed as a probability. *See also:* **error tolerance; fault tolerance; robustness**.

back-to-back testing. Testing in which two or more variants of a program are executed with the same inputs, the outputs are compared, and errors are analyzed in case of discrepancies. *See also:* **mutation testing**.

background. In job scheduling, the computing environment in which low-priority processes or those not requiring user interaction are executed. *Contrast with:* **foreground**. *See also:* **background processing**.

background processing. The execution of a low-priority process while higher priority processes are not using computer resources, or the execution of processes that do not require user interaction. *Contrast with:* **foreground processing**.

backup. (1) A system, component, file, procedure, or person available to replace or help restore a primary item in the event of a failure or externally caused disaster.

(2) To create or designate a system, component, file, procedure, or person as in (1).

backup programmer. The assistant leader of a chief programmer team; responsibilities include contributing significant portions of the software being developed by the team, aiding the chief programmer in reviewing the work of other team members, substituting for the chief programmer when

necessary, and having an overall technical understanding of the software being developed. *See also:* **chief programmer.**

backward execution. *See:* **reversible execution.**

backward recovery. (1) The reconstruction of a file to a given state by reversing all changes made to the file since it was in that state.

(2) A type of recovery in which a system, program, database, or other system resource is restored to a previous state in which it can perform required functions.

Contrast with: **forward recovery.**

base address. An address used as a reference point to which a relative address is added to determine the address of the storage location to be accessed. *See also:* **indexed address; relative address; self-relative address.**

baseline. (1) A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

(2) A document or a set of such documents formally designated and fixed at a specific time during the life cycle of a configuration item. *Note:* Baselines, plus approved changes from those baselines, constitute the current configuration identification. *See also:* **allocated baseline; developmental configuration; functional baseline; product baseline.**

(3) Any agreement or result designated and fixed at a given time, from which changes require justification and approval.

baseline management. In configuration management, the application of technical and administrative direction to designate the documents and changes to those documents that formally identify and establish baselines at specific times during the life cycle of a configuration item.

batch. Pertaining to a system or mode of operation in which inputs are collected and processed all at one time, rather than being processed as they arrive, and a job, once started, proceeds to completion without

additional input or user interaction. *Contrast with:* **conversational; interactive; online; real time.**

bathtub curve. A graph of the number of failures in a system or component as a function of time. The name is derived from the usual shape of the graph: a period of decreasing failures (the early-failure period), followed by a relatively steady period (the constant-failure period), followed by a period of increasing failures (the wearout-failure period).

benchmark. (1) A standard against which measurements or comparisons can be made.

(2) A procedure, problem, or test that can be used to compare systems or components to each other or to a standard as in (1).

(3) A recovery file.

big-bang testing. A type of integration testing in which software elements, hardware elements, or both are combined all at once into an overall system, rather than in stages.

binary digit (bit). (1) A unit of information that can be represented by either a zero or a one.

(2) An element of computer storage that can hold a unit of information as in (1).

(3) A numeral used to represent one of the two digits in the binary numeration system; zero (0) or one (1).

See also: **byte; word.**

bind. To assign a value to an identifier. For example, to assign a value to a parameter or to assign an absolute address to a symbolic address in a computer program. *See also:* **dynamic binding; static binding.**

bit. Acronym for **binary digit.**

bit steering. A microprogramming technique in which the meaning of a field in a microinstruction is dependent on the value of another field in the microinstruction. *Syn:* **immediate control.** *Contrast with:* **residual control.** *See also:* **two-level encoding.**

black box. (1) A system or component whose inputs, outputs, and general function are

known but whose contents or implementation are unknown or irrelevant. *Contrast with: glass box.*

(2) Pertaining to an approach that treats a system or component as in (1). *See also: encapsulation.*

black-box testing. *See: functional testing (1).*

block. (1) A group of contiguous storage locations, computer program statements, records, words, characters, or bits that are treated as a unit. *See also: block-structured language; delimiter.*

(2) To form a group as in (1). *Contrast with: deblock.*

block allocation. *See: paging (1).*

block diagram. A diagram of a system, computer, or device in which the principal parts are represented by suitably annotated geometrical figures to show both the functions of the parts and their functional relationships. *Syn: configuration diagram; system resources chart. See also: box diagram; bubble chart; flowchart; graph; input-process-output chart; structure chart.*

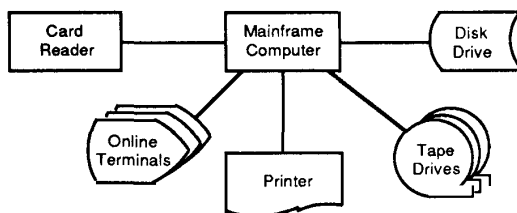


Fig 1
Block Diagram

block-structured language. A design or programming language in which sequences of statements, called blocks, are defined, usually with begin and end delimiters, and variables or labels defined in one block are not recognized outside that block. Examples include Ada, ALGOL, PL/I. *See also: structured programming language.*

blocking factor. The number of records, words, characters, or bits in a block.

boot. To initialize a computer system by clearing memory and reloading the operating system. Derived from **bootstrap**.

bootstrap. (1) A short computer program that is permanently resident or easily loaded into a computer and whose execution brings a larger program, such as an operating system or its loader, into memory.

(2) To use a program as in (1). *Syn: initial program load.*

bootstrap loader. A short computer program used to load a bootstrap.

bottom-up. Pertaining to an activity that starts with the lowest-level components of a hierarchy and proceeds through progressively higher levels; for example, bottom-up design; bottom-up testing. *Contrast with: top-down. See also: critical piece first.*

boundary value. A data value that corresponds to a minimum or maximum input, internal, or output value specified for a system or component. *See also: stress testing.*

box diagram. A control flow diagram consisting of a rectangle that is subdivided to show sequential steps, if-then-else conditions, repetition, and case conditions. *Syn: Chapin chart; Nassi-Shneiderman chart; program structure diagram. See also: block diagram; bubble chart; flowchart; graph; input-process-output chart; program structure diagram; structure chart.*

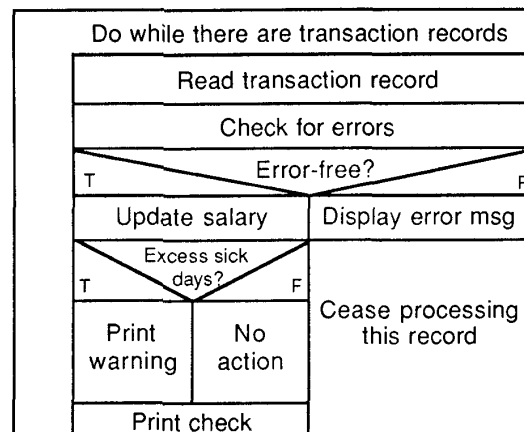


Fig 2
Box Diagram

branch. (1) A computer program construct in which one of two or more alternative sets of

program statements is selected for execution. *See also:* **case; jump; go to; if-then-else.**

(2) A point in a computer program at which one of two or more alternative sets of program statements is selected for execution. *Syn:* **branchpoint.**

(3) Any of the alternative sets of program statements in (1).

(4) To perform the selection in (1).

branch testing. Testing designed to execute each outcome of each decision point in a computer program. *Contrast with:* **path testing; statement testing.**

branchpoint. *See:* **branch** (2).

breakpoint. A point in a computer program at which execution can be suspended to permit manual or automated monitoring of program performance or results. Types include code breakpoint, data breakpoint, dynamic breakpoint, epilog breakpoint, program-mable breakpoint, prolog breakpoint, static breakpoint. *Note:* A breakpoint is said to be set when both a point in the program and an event that will cause suspension of execution at that point are defined; it is said to be initiated when program execution is suspended.

bubble chart. A data flow, data structure, or other diagram in which entities are depicted with circles (bubbles) and relationships are represented by links drawn between the circles. *See also:* **block diagram; box diagram; flowchart; graph; input-process-output chart; structure chart.**

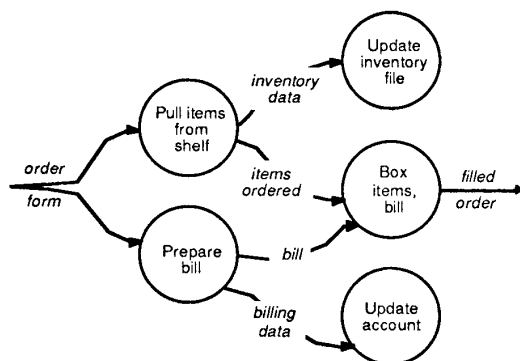


Fig 3
Bubble Chart

buffer. (1) A device or storage area used to store data temporarily to compensate for differences in rates of data flow, time of occurrence of events, or amounts of data that can be handled by the devices or processes involved in the transfer or use of the data.

(2) A routine that accomplishes the objectives in (1).

(3) To allocate, schedule, or use devices or storage areas as in (1). *See also:* **anticipatory buffering; dynamic buffering; simple buffering.**

bug. *See:* **error; fault.**

bug seeding. *See:* **error seeding.**

build. An operational version of a system or component that incorporates a specified subset of the capabilities that the final product will provide.

burn-in period. *See:* **early-failure period.**

busy. Pertaining to a system or component that is operational, in service, and in use. *See also:* **down; idle; up.**

busy time. In computer performance engineering, the period of time during which a system or component is operational, in service, and in use. *See also:* **down time; idle time; set-up time; up time.**

byte. (1) A group of adjacent binary digits operated upon as a unit and usually shorter than a computer word (frequently connotes a group of eight bits).

(2) An element of computer storage that can hold a group of bits as in (1).

See also: **bit; word.**

call. (1) A transfer of control from one software module to another, usually with the implication that control will be returned to the calling module. *Contrast with:* **go to.**

(2) A computer instruction that transfers control from one software module to another as in (1) and, often, specifies the parameters to be passed to and from the module.

(3) To transfer control from one software module to another as in (1) and, often, to pass

parameters to the other module. *Syn:* cue.

See also: call by name; call by reference; call by value; call list; calling sequence.

call by address. *See:* call by reference.

call by location. *See:* call by reference.

call by name. A method for passing parameters, in which the calling module provides to the called module a symbolic expression representing the parameter to be passed, and a service routine evaluates the expression and provides the resulting value to the called module. *Note:* Because the expression is evaluated each time its corresponding formal parameter is used in the called module, the value of the parameter may change during the execution of the called module. *Contrast with:* call by reference; call by value.

call by reference. A method for passing parameters, in which the calling module provides to the called module the address of the parameter to be passed. *Note:* With this method, the called module has the ability to change the value of the parameter stored by the calling module. *Syn:* call by address; call by location. *Contrast with:* call by name; call by value.

call by value. A method of passing parameters, in which the calling module provides to the called module the actual value of the parameter to be passed. *Note:* With this method, the called module cannot change the value of the parameter as stored by the calling module. *Contrast with:* call by name; call by reference.

call graph. A diagram that identifies the modules in a system or computer program and shows which modules call one another. *Note:* The result is not necessarily the same as that shown in a structure chart. *Syn:* call tree; tier chart. *Contrast with:* structure chart. *See also:* control flow diagram; data flow diagram; data structure diagram; state diagram.

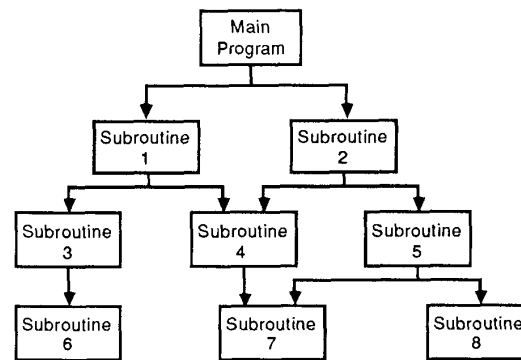


Fig 4
Call Graph

call list. The ordered list of arguments used in a call to a software module.

call trace. *See:* subroutine trace.

call tree. *See:* call graph.

calling sequence. A sequence of computer instructions and, possibly, data necessary to perform a call to another module.

CASE. Acronym for **computer-aided software engineering**.

case. A single-entry, single-exit multiple-way branch that defines a control expression, specifies the processing to be performed for each value of the control expression, and returns control in all instances to the statement immediately following the overall construct. *Syn:* multiple exclusive selective construct. *Contrast with:* go to; jump; if-then-else. *See also:* multiple inclusive selective construct.

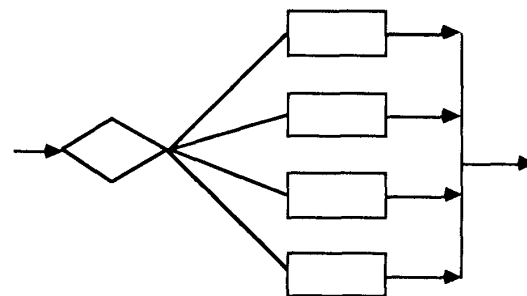


Fig 5
Case Construct

catastrophic failure. A failure of critical software.

CCB. (1) Acronym for **configuration control board**.

(2) Acronym for **change control board**. *See: configuration control board*.

CDR. Acronym for **critical design review**.

certification. (1) A written guarantee that a system or component complies with its specified requirements and is acceptable for operational use. For example, a written authorization that a computer system is secure and is permitted to operate in a defined environment.

(2) A formal demonstration that a system or component complies with its specified requirements and is acceptable for operational use.

(3) The process of confirming that a system or component complies with its specified requirements and is acceptable for operational use.

change control. *See: configuration control*.

change control board. *See: configuration control board*.

change dump. A selective dump of those storage locations whose contents have changed since some specified time or event. *Syn: differential dump. See also: dynamic dump; memory dump; postmortem dump; selective dump; snapshot dump; static dump.*

channel capacity. The maximum amount of information that can be transferred on a given channel per unit of time; usually measured in bits per second or in baud. *See also: memory capacity; storage capacity.*

Chapin chart. *See: box diagram*.

character. A letter, digit, or other symbol that is used to represent information.

character type. A data type whose members can assume the values of specified characters and can be operated on by character operators, such as concatenation.

Contrast with: enumeration type; integer type; logical type; real type.

characteristic. (IEEE Std 1008-1987 [10]) *See: data characteristic; software characteristic.*

checkout. Testing conducted in the operational or support environment to ensure that a software product performs as required after installation.

checkpoint. A point in a computer program at which program state, status, or results are checked or recorded.

chief programmer. The leader of a chief programmer team; a senior-level programmer whose responsibilities include producing key portions of the software assigned to the team, coordinating the activities of the team, reviewing the work of the other team members, and having an overall technical understanding of the software being developed. *See also: backup programmer; chief programmer team.*

chief programmer team. A software development group that consists of a chief programmer, a backup programmer, a secretary/librarian, and additional programmers and specialists as needed, and that employs procedures designed to enhance group communication and to make optimum use of each member's skills. *See also: backup programmer; chief programmer; egoless programming.*

CI. Acronym for **configuration item**.

clear. To set a variable, register, or other storage location to zero, blank, or other null value. *See also: initialize; reset.*

closed loop. A loop that has no exit and whose execution can be interrupted only by intervention from outside the computer program or procedure in which the loop is located. *Contrast with: UNTIL; WHILE.*

closed subroutine. A subroutine that is stored at one given location rather than being copied into a computer program at each place that it is called. *Contrast with: open subroutine.*

CM. Acronym for **configuration management**.

code. (1) In software engineering, computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler, or other translator. *See also:* **source code; object code; machine code; microcode.**

(2) To express a computer program in a programming language.

(3) A character or bit pattern that is assigned a particular meaning; for example, a status code.

code breakpoint. A breakpoint that is initiated upon execution of a given computer instruction. *Syn:* **control breakpoint.** *Contrast with:* **data breakpoint.** *See also:* **dynamic breakpoint; epilog breakpoint; programmable breakpoint; prolog breakpoint; static breakpoint.**

code generator. (1) A routine, often part of a compiler, that transforms a computer program from some intermediate level of representation (often the output of a root compiler or parser) into a form that is closer to the language of the machine on which the program will execute.

(2) A software tool that accepts as input the requirements or design for a computer program and produces source code that implements the requirements or design. *Syn:* **source code generator.** *See also:* **application generator.**

code inspection. *See:* **inspection.**

code of ethics standard. (IEEE Std 1002-1987 [9]) A standard that describes the characteristics of a set of moral principles dealing with accepted standards of conduct by, within, and among professionals.

code review. A meeting at which software code is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. *Contrast with:* **design review; formal qualification review; requirements review; test readiness review.**

code trace. *See:* **execution trace.**

coding. (1) In software engineering, the process of expressing a computer program in a programming language.

(2) (IEEE Std 1002-1987 [9]) The transforming of logic and data from design specifications (design descriptions) into a programming language.

See also: **software development process.**

cohesion. The manner and degree to which the tasks performed by a single software module are related to one another. Types include coincidental, communicational, functional, logical, procedural, sequential, and temporal. *Syn:* **module strength.** *Contrast with:* **coupling.**

coincidental cohesion. A type of cohesion in which the tasks performed by a software module have no functional relationship to one another. *Contrast with:* **communicational cohesion; functional cohesion; logical cohesion; procedural cohesion; sequential cohesion; temporal cohesion.**

command. An expression that can be input to a computer system to initiate an action or affect the execution of a computer program; for example, the "log on" command to initiate a computer session.

command-driven. Pertaining to a system or mode of operation in which the user directs the system through commands. *Contrast with:* **menu-driven.**

command language. A language used to express commands to a computer system. *See also:* **command-driven.**

comment. Information embedded within a computer program, job control statements, or a set of data, that provides clarification to human readers but does not affect machine interpretation.

common. *See:* **common storage.**

common area. *See:* **common storage.**

common block. *See:* **common storage.**

common coupling. *See:* **common-environment coupling.**

common data. *See:* **global data.**

common-environment coupling. A type of coupling in which two software modules access a common data area. *Syn:* **common coupling.** *Contrast with:* **content coupling; control coupling; data coupling; hybrid coupling; pathological coupling.**

common storage. A portion of main storage that can be accessed by two or more modules in a software system. *Syn:* **common area; common block.** *See also:* **global data.**

communicational cohesion. A type of cohesion in which the tasks performed by a software module use the same input data or contribute to producing the same output data. *Contrast with:* **coincidental cohesion; functional cohesion; logical cohesion; procedural cohesion; sequential cohesion; temporal cohesion.**

compaction. In microprogramming, the process of converting a microprogram into a functionally equivalent microprogram that is faster or shorter than the original. *See also:* **local compaction; global compaction.**

comparator. A software tool that compares two computer programs, files, or sets of data to identify commonalities or differences. Typical objects of comparison are similar versions of source code, object code, data base files, or test results.

compatibility. (1) The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment.
(2) The ability of two or more systems or components to exchange information.
See also: **interoperability.**

compile. To translate a computer program expressed in a high order language into its machine language equivalent. *Contrast with:* **assemble; decompile; interpret.**

compile-and-go. An operating technique in which there are no stops between the compiling, linking, loading, and execution of a computer program.

compiler. A computer program that translates programs expressed in a high order language into their machine language equivalents. *Contrast with:* **assembler; interpreter.** *See also:* **cross-compiler; incremental compiler; root compiler.**

compiler code. Computer instructions and data definitions expressed in a form that can be recognized and processed by a compiler. *Contrast with:* **assembly code; interpretive code; machine code.**

compiler compiler. *See:* **compiler generator.**

compiler generator. A translator or interpreter used to construct part or all of a compiler. *Syn:* **compiler compiler; meta-compiler.**

completion code. A code communicated to a job stream processor by a batch program to influence the execution of succeeding steps in the input stream.

complexity. (1) The degree to which a system or component has a design or implementation that is difficult to understand and verify. *Contrast with:* **simplicity.**
(2) Pertaining to any of a set of structure-based metrics that measure the attribute in (1).

component. One of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. *Note:* The terms "module," "component," and "unit" are often used interchangeably or defined to be subelements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized.

component standard. (IEEE Std 1002-1987 [9])
A standard that describes the characteristics of data or program components.

component testing. Testing of individual hardware or software components or groups of related components. *Syn:* **module testing.** *See also:* **integration testing; interface testing; system testing; unit testing.**

composite type. A data type each of whose members is composed of multiple data items. For example, a data type called PAIRS whose members are ordered pairs (x,y) . *Contrast with:* **atomic type**.

computer-aided software engineering (CASE). The use of computers to aid in the software engineering process. May include the application of software tools to software design, requirements tracing, code production, testing, document generation, and other software engineering activities.

computer instruction. (1) A statement in a programming language, specifying an operation to be performed by a computer and the addresses or values of the associated operands; for example, Move A to B. *See also:* **instruction format; instruction set**.
(2) Loosely, any executable statement in a computer program.

computer language. A language designed to enable humans to communicate with computers. *See also:* **design language; query language; programming language**. *Note:* P610.13 [17] defines specific computer languages.

computer performance evaluation. An engineering discipline that measures the performance of computer systems and investigates methods by which that performance can be improved. *See also:* **system profile; throughput; utilization; workload model**.

computer program. A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions. *See also:* **software**.

computer program abstract. A brief description of a computer program that provides sufficient information for potential users to determine the appropriateness of the program to their needs and resources.

computer program component (CPC).* *See:* **computer software component**.
*Deprecated.

computer program configuration item (CPCI).* *See:* **computer software configuration item**.

*Deprecated.

computer resource allocation. The assignment of computer resources to current and waiting jobs; for example, the assignment of main memory, input/output devices, and auxiliary storage to jobs executing concurrently in a computer system. *See also:* **dynamic resource allocation; storage allocation**.

computer resources. The computer equipment, programs, documentation, services, facilities, supplies, and personnel available for a given purpose. *See also:* **computer resource allocation**.

computer security. *Note:* P610.9 [16] defines terminology pertaining to computer security.

computer software component (CSC). A functionally or logically distinct part of a computer software configuration item, typically an aggregate of two or more software units.

computer software configuration item (CSCI). An aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process. *Contrast with:* **hardware configuration item**. *See also:* **configuration item**.

computer system. A system containing one or more computers and associated software.

computer word. *See:* **word**.

computing center. A facility designed to provide computer services to a variety of users through the operation of computers and auxiliary hardware and through services provided by the facility's staff.

concept phase. (1) (IEEE Std 1002-1987 [9]) The period of time in the software development cycle during which the user needs are described and evaluated through documentation (for example, statement of needs,

advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project).

(2) (IEEE Std 1012-1986 [12]) The initial phase of a software development project, in which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project).

concurrent. Pertaining to the occurrence of two or more activities within the same interval of time, achieved either by interleaving the activities or by simultaneous execution. *Syn:* **parallel (2)**. *Contrast with:* **simultaneous**.

condition code. *See:* **status code**.

conditional branch.* *See:* **conditional jump**.
*Deprecated.

conditional jump. A jump that takes place only when specified conditions are met. *Contrast with:* **unconditional jump**.

configuration. (1) The arrangement of a computer system or component as defined by the number, nature, and interconnections of its constituent parts.

(2) In configuration management, the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product. *See also:* **configuration item; form, fit and function; version**.

configuration audit. *See:* **functional configuration audit; physical configuration audit**.

configuration control. An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification. *Syn:* **change control**. *Contrast with:* **configuration identification; configuration status accounting**. *See also:* **configuration control board; deviation; engineering change;**

interface control; notice of revision; specification change notice; waiver.

configuration control board (CCB). A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. *Syn:* **change control board**. *See also:* **configuration control**.

configuration diagram. *See:* **block diagram**.

configuration identification. (1) An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation. *Contrast with:* **configuration control; configuration status accounting**.

(2) The current approved technical documentation for a configuration item as set forth in specifications, drawings, associated lists, and documents referenced therein. *See also:* **allocated configuration identification; functional configuration identification; product configuration identification; baseline**.

configuration index. A document used in configuration management, providing an accounting of the configuration items that make up a product. *See also:* **configuration item development record; configuration status accounting**.

configuration item (CI). An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process. *See also:* **hardware configuration item; computer software configuration item; configuration identification; critical item**.

configuration item development record. A document used in configuration management, describing the development status of a configuration item based on the results of configuration audits and design reviews. *See also:* **configuration index; configuration status accounting**.

configuration management (CM). A discipline applying technical and administra-

tive direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. *See also:* **baseline; configuration identification; configuration control; configuration status accounting; configuration audit.**

configuration status accounting. An element of configuration management, consisting of the recording and reporting of information needed to manage a configuration effectively. This information includes a listing of the approved configuration identification, the status of proposed changes to the configuration, and the implementation status of approved changes. *Contrast with:* **configuration control; configuration identification.** *See also:* **configuration index; configuration item development record.**

consecutive. Pertaining to the occurrence of two sequential events or items without the intervention of any other event or item; that is, one immediately after the other.

consistency. The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component. *See also:* **traceability.**

constant. A quantity or data item whose value cannot change; for example, the data item FIVE, with an unchanging value of 5. *Contrast with:* **variable.** *See also:* **figurative constant; literal.**

constant-failure period. The period of time in the life cycle of a system or component during which hardware failures occur at an approximately uniform rate. *Contrast with:* **early-failure period; wearout-failure period.** *See also:* **bathtub curve.**

content coupling. A type of coupling in which some or all of the contents of one software module are included in the contents of another module. *Contrast with:* **common-environment coupling; control coupling; data coupling; hybrid coupling; pathological coupling.**

contiguous allocation. A storage allocation technique in which programs or data to be stored are allocated a block of storage of equal or greater size, so that logically contiguous programs and data are assigned physically contiguous storage locations. *Contrast with:* **paging (1).**

continuous iteration. A loop that has no exit.

control breakpoint. *See:* **code breakpoint.**

control coupling. A type of coupling in which one software module communicates information to another module for the explicit purpose of influencing the latter module's execution. *Contrast with:* **common-environment coupling; content coupling; data coupling; hybrid coupling; pathological coupling.**

control data. Data that select an operating mode, direct the sequential flow of a program, or otherwise directly influence the operation of software; for example, a loop control variable.

control flow. The sequence in which operations are performed during the execution of a computer program. *Syn:* **flow of control.** *Contrast with:* **data flow.**

control flow diagram. A diagram that depicts the set of all possible sequences in which operations may be performed during the execution of a system or program. Types include box diagram, flowchart, input-process-output chart, state diagram. *Contrast with:* **data flow diagram.** *See also:* **call graph; structure chart.**

control flow trace. *See:* **execution trace.**

control language. *See:* **job control language.**

control program. *See:* **supervisory program.**

control statement. A program statement that selects among alternative sets of program statements or affects the order in which operations are performed. For example, if-then-else, case. *Contrast with:* **assignment statement; declaration.**

control store. In a microprogrammed computer, the computer memory in which microprograms reside. *See also:* **microword**; **nanostore**.

control variable. *See:* **loop-control variable**.

conventions. (IEEE Std 983-1986 [7]) Requirements employed to prescribe a disciplined uniform approach to providing consistency in a software product, that is, uniform patterns or forms for arranging data. *See also:* **practices**; **standards**.

conversational. Pertaining to an interactive system or mode of operation in which the interaction between the user and the system resembles a human dialog. *Contrast with:* **batch**. *See also:* **interactive**; **on-line**; **real time**.

conversational compiler. *See:* **incremental compiler**.

conversion. Modification of existing software to enable it to operate with similar functional capability in a different environment; for example, converting a program from Fortran to Ada, converting a program that runs on one computer to run on another.

copy. (1) To read data from a source, leaving the source data unchanged, and to write the same data elsewhere in a physical form that may differ from that of the source. For example, to copy data from a magnetic disk onto a magnetic tape. *Contrast with:* **move**.
(2) The result of a copy process as in (1). For example, a copy of a data file.

core dump.* *See:* **memory dump**.
*Deprecated.

coroutine. A routine that begins execution at the point at which operation was last suspended, and that is not required to return control to the program or subprogram that called it. *Contrast with:* **subroutine**.

corrective maintenance. Maintenance performed to correct faults in hardware or software. *Contrast with:* **adaptive maintenance**; **perfective maintenance**.

correctness. (1) The degree to which a system or component is free from faults in its specification, design, and implementation.

(2) The degree to which software, documentation, or other items meet specified requirements.

(3) The degree to which software, documentation, or other items meet user needs and expectations, whether specified or not.

correctness proof. *See:* **proof of correctness**.

counter. A variable used to record the number of occurrences of a given event during the execution of a computer program; for example, a variable that records the number of times a loop is executed.

coupling. The manner and degree of interdependence between software modules. Types include common-environment coupling, content coupling, control coupling, data coupling, hybrid coupling, and pathological coupling. *Contrast with:* **cohesion**.

CPC. Acronym for **computer program component**. *See:* **computer software component**.

CPCI. Acronym for **computer program configuration item**. *See:* **computer software configuration item**.

crash. The sudden and complete failure of a computer system or component. *See also:* **hard failure**.

critical design review (CDR). (1) A review conducted to verify that the detailed design of one or more configuration items satisfy specified requirements; to establish the compatibility among the configuration items and other items of equipment, facilities, software, and personnel; to assess risk areas for each configuration item; and, as applicable, to assess the results of producibility analyses, review preliminary hardware product specifications, evaluate preliminary test planning, and evaluate the adequacy of preliminary operation and support documents. *See also:* **preliminary design review**; **system design review**.

(2) A review as in (1) of any hardware or software component.

critical item. In configuration management, an item within a configuration item that, because of special engineering or logistic considerations, requires an approved specification to establish technical or inventory control at the component level.

critical piece first. A system development approach in which the most critical aspects of a system are implemented first. The critical piece may be defined in terms of services provided, degree of risk, difficulty, or other criteria. *See also:* **bottom-up; top-down.**

critical software. (IEEE Std 1012-1986 [12]) Software whose failure could have an impact on safety, or could cause large financial or social loss.

criticality. The degree of impact that a requirement, module, error, fault, failure, or other item has on the development or operation of a system. *Syn:* **severity.**

cross-assembler. An assembler that executes on one computer but generates machine code for a different computer.

cross-compiler. A compiler that executes on one computer but generates machine code for a different computer.

cross-reference generator. A software tool that accepts as input the source code of a computer program and produces as output a listing that identifies each of the program's variables, labels, and other identifiers and indicates which statements in the program define, set, or use each one. *Syn:* **cross-referencer.**

cross-reference list. A list that identifies each of the variables, labels, and other identifiers in a computer program and indicates which statements in the program define, set, or use each one.

cross-referencer. *See:* **cross-reference generator.**

CSC. Acronym for **computer software component.**

CSCI. Acronym for **computer software configuration item.**

cue. *See:* **call (3).**

curriculum standard. (IEEE Std 1002-1987 [9]) A standard that describes the characteristics of a course of study on a body of knowledge that is offered by an educational institution.

cycle. (1) A period of time during which a set of events is completed. *See also:* **software development cycle; software life cycle.**

(2) A set of operations that is repeated regularly in the same sequence, possibly with variations in each repetition; for example, a computer's read cycle. *See also:* **pass.**

cycle stealing. The process of suspending the operation of a central processing unit for one or more cycles to permit the occurrence of other operations, such as transferring data from main memory in response to an output request from an input/output controller.

cyclic search. A storage allocation technique in which each search for a suitable block of storage begins with the block following the one last allocated.

data. (1) A representation of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means. *See also:* **data type.** *Note:* IEEE Std 610.5-1990 [2] defines terminology pertaining to data management.
(2) Sometimes used as a synonym for documentation.

data abstraction. (1) The process of extracting the essential characteristics of data by defining data types and their associated functional characteristics and disregarding representation details. *See also:* **encapsulation; information hiding.**
(2) The result of the process in (1).

data breakpoint. A breakpoint that is initiated when a specified data item is accessed. *Syn:* **storage breakpoint.** *Contrast with:* **code breakpoint.** *See also:* **dynamic breakpoint; epilog breakpoint; programmable breakpoint; prolog breakpoint; static breakpoint.**

data characteristic. (IEEE Std 1008-1987 [10]) An inherent, possibly accidental, trait,

quality, or property of data (for example, arrival rates, formats, value ranges, or relationships between field values).

data coupling. A type of coupling in which output from one software module serves as input to another module. *Syn:* **input-output coupling**. *Contrast with:* **common-environment coupling; content coupling; control coupling; hybrid coupling; pathological coupling**.

data definition. *Note:* This term is defined in IEEE Std 610.5-1990 [2].

data exception. An exception that occurs when a program attempts to use or access data incorrectly. *See also:* **addressing exception; operation exception; overflow exception; protection exception; underflow exception**.

data flow. The sequence in which data transfer, use, and transformation are performed during the execution of a computer program. *Contrast with:* **control flow**.

data flow diagram (DFD). A diagram that depicts data sources, data sinks, data storage, and processes performed on data as nodes, and logical flow of data as links between the nodes. *Syn:* **data flowchart; data flow graph**. *Contrast with:* **control flow diagram; data structure diagram**.

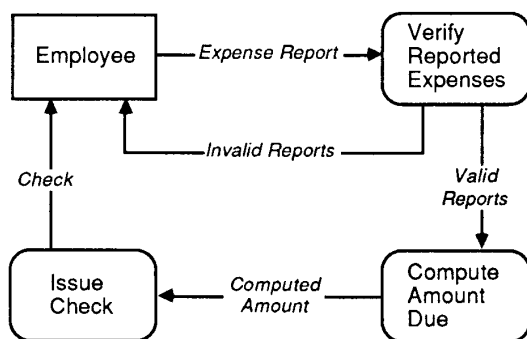


Fig 6
Data Flow Diagram

data flow graph. *See:* **data flow diagram**.

data flow trace. *See:* **variable trace**.

data flowchart (flow chart). *See:* **data flow diagram**.

data input sheet. User documentation that describes, in a worksheet format, the required and optional input data for a system or component. *See also:* **user manual**.

data-sensitive fault. A fault that causes a failure in response to some particular pattern of data. *Syn:* **pattern-sensitive fault**. *Contrast with:* **program-sensitive fault**.

data structure. A physical or logical relationship among data elements, designed to support specific data manipulation functions. *Note:* IEEE Std 610.5-1990 [2] defines specific data structures.

data structure-centered design. A software design technique in which the architecture of a system is derived from analysis of the structure of the data sets with which the system must deal. *See also:* **input-process-output; modular decomposition; object-oriented design; rapid prototyping; stepwise refinement; structure clash; structured design; transaction analysis; transform analysis**.

data structure diagram. A diagram that depicts a set of data elements, their attributes, and the logical relationships among them. *Contrast with:* **data flow diagram**. *See also:* **entity-relationship diagram**.

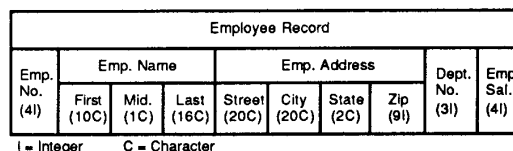


Fig 7
Data Structure Diagram

data trace. *See:* **variable trace**.

data type. A class of data, characterized by the members of the class and the operations that can be applied to them. For example, character type, enumeration type, integer type, logical type, real type. *See also:* **strong typing**.

database. A collection of interrelated data stored together in one or more computerized files. *Note:* IEEE Std 610.5-1990 [2] defines terminology pertaining to databases.

datum. Singular for **data**.

deadlock. A situation in which computer processing is suspended because two or more devices or processes are each awaiting resources assigned to the others. *See also:* **lockout**.

deassembler.* *See:* **disassembler**.
* Deprecated.

deblock. To separate the parts of a block.
Contrast with: **block** (2).

debug. To detect, locate, and correct faults in a computer program. Techniques include use of breakpoints, desk checking, dumps, inspection, reversible execution, single-step operation, and traces.

decision table. A table used to show sets of conditions and the actions resulting from them.

declaration. A non-executable program statement that affects the assembler or compiler's interpretation of other statements in the program. For example, a statement that identifies a name, specifies what the name represents, and, possibly, assigns it an initial value. *Contrast with:* **assignment statement; control statement**. *See also:* **pseudo instruction**.

declarative language. A nonprocedural language that permits the user to declare a set of facts and to express queries or problems that use these facts. *See also:* **interactive language; rule-based language**.

decompile. To translate a compiled computer program from its machine language version into a form that resembles, but may not be identical to, the original high order language program. *Contrast with:* **compile**.

decompiler. A software tool that decompiles computer programs.

decoupling. The process of making software modules more independent of one another to decrease the impact of changes to, and errors in, the individual modules. *See also:* **coupling**.

delimiter. A character or set of characters used to denote the beginning or end of a group of related bits, characters, words, or statements.

delivery. Release of a system or component to its customer or intended user. *See also:* **software life cycle; system life cycle**.

demand paging. A storage allocation technique in which pages are transferred from auxiliary storage to main storage only when those pages are needed. *Contrast with:* **anticipatory paging**.

demodularization. In software design, the process of combining related software modules, usually to optimize system performance. *See also:* **downward compression; lateral compression; upward compression**.

demonstration. A dynamic analysis technique that relies on observation of system or component behavior during execution, without need for post-execution analysis, to detect errors, violations of development standards, and other problems. *See also:* **testing**.

derived type. A data type whose members and operations are taken from those of another data type according to some specified rule. *See also:* **subtype**.

description standard. (IEEE Std 1002-1987 [9]) A standard that describes the characteristics of product information or procedures provided to help understand, test, install, operate, or maintain the product.

design. (1) The process of defining the architecture, components, interfaces, and other characteristics of a system or component. *See also:* **architectural design; preliminary design; detailed design**.
(2) The result of the process in (1).

design description. A document that describes the design of a system or component.

Typical contents include system or component architecture, control logic, data structures, input/output formats, interface descriptions, and algorithms. *Syn:* **design document; design specification**. *See also:* **product specification**. *Contrast with:* **requirements specification**.

design document. *See:* **design description**.

design element. (IEEE Std 990-1987 [8]) A basic component or building block in a design.

design entity. (IEEE Std 1016-1987 [13]) An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.

design inspection. *See:* **inspection**.

design language. A specification language with special constructs and, sometimes, verification protocols, used to develop, analyze, and document a hardware or software design. Types include hardware design language, program design language. *See also:* **requirements specification language**.

design level. (IEEE Std 829-1983 [5]) The design decomposition of the software item (for example, system, subsystem, program, or module).

design phase. The period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements. *See also:* **detailed design; preliminary design**.

design requirement. A requirement that specifies or constrains the design of a system or system component. *Contrast with:* **functional requirement; implementation requirement; interface requirement; performance requirement; physical requirement**.

design review. A process or meeting during which a system, hardware, or software design is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include critical design review, preliminary

design review, system design review. *Contrast with:* **code review; formal qualification review; requirements review; test readiness review**.

design specification. *See:* **design description**.

design standard. (IEEE Std 1002-1987 [9]) A standard that describes the characteristics of a design or a design description of data or program components.

design unit. (IEEE Std 990-1987 [8]) A logically related collection of design elements. In an Ada PDL, a design unit is represented by an Ada compilation unit.

design view. (IEEE Std 1016-1987 [13]) A subset of design entity attribute information that is specifically suited to the needs of a software project activity.

desk checking. A static analysis technique in which code listings, test results, or other documentation are visually examined, usually by the person who generated them, to identify errors, violations of development standards, or other problems. *See also:* **inspection; walk-through**.

destination address. The address of the device or storage location to which data is to be transferred. *Contrast with:* **source address**.

destructive read. A read operation that alters the data in the accessed location. *Contrast with:* **nondestructive read**.

detailed design. (1) The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented. *See also:* **software development process**.

(2) The result of the process in (1).

development cycle. *See:* **software development cycle**.

development life cycle. *See:* **software development cycle**.

development specification. *See:* **requirements specification**.